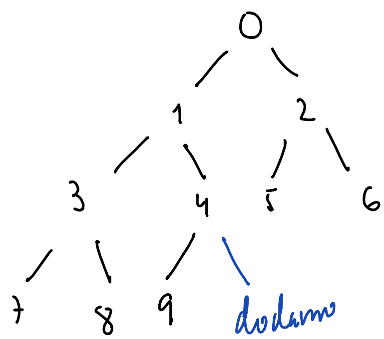


V tabeli a

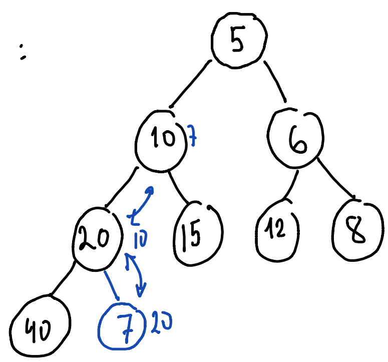
a[i]

a[2i+1] a[2i+2] sinova

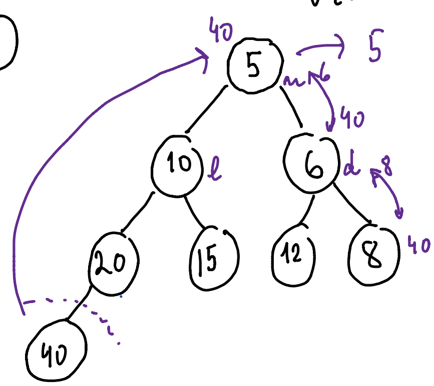
a[(j-1)//2] oče

a[j]

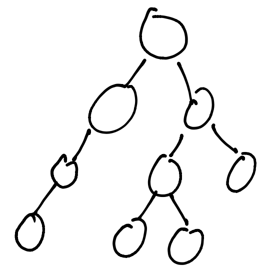
Dodaj:



Vzemi:

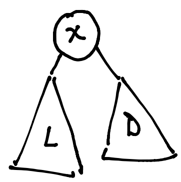


Dvojiška drevesa



Rekurzivna podatkovna struktura:

- prazno drevo je dvojiško drevo
- če sta L in D dvojiški drevesi, je tudi sestavljeno drevo



dvojiško drevo.

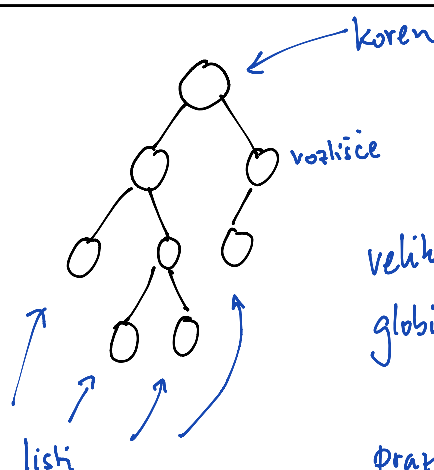
Seznami so rekurzivni:

- prazen seznam []
- element x, seznam r

⇒ seznam $x \to r$

$5 \to 3 \to []$

8



velikost = # vozlišč

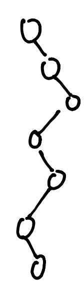
globina = 4 (najdaljša pot od korena do lista)

prazno drevo ima globino 0

V kakšni zvezi sta velikost in globina?

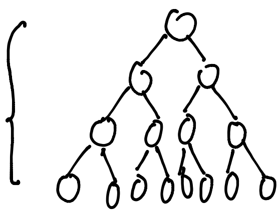
velikost = n

globina = g



$$\log_2 n \leq g \leq n$$

off-by-one

g 

polno drevo : $n = 1 + 2 + 4 + \dots + 2^g = 2^{g+1} - 1$

$$n \approx 2^g$$

$$g \approx \log_2 n$$

$$g = \log(n+1) - 1$$

$$\log n$$

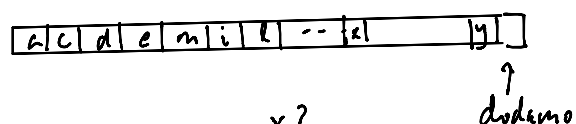
Iskalna drevesa

Podatkovna struktura:

- dodaj x
- poišči x
- odstrani x

Naivna rešitev: tabela

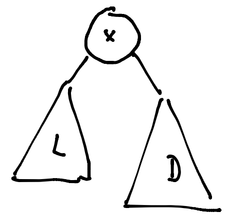
- dodamo na konec $O(1)$
- poiščemo; pregledamo tabelo $O(n)$
- odstrani x: poišči indeks x in ga nadomesti z zadnjim v tabeli $O(n)$



Urejena tabela:

- poišči (bisekcija): $\log n \in O(\log n)$
- dodaj: $\log n + n \in O(n)$
- odstrani: $\log n + n \in O(n)$

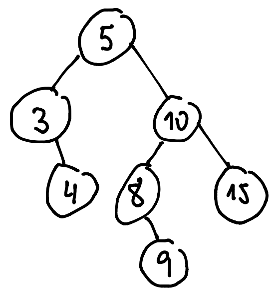
Iskalno drevo:



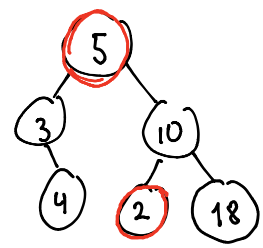
- → dvojiško drevo
- → vsi elementi L so manjši od x
- → vsi elementi D so večji od x
- → L in D sta iskalni drevesi

$L < x < D$

Primer:

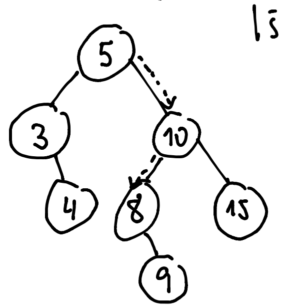


Protiprimer:



Kako iščemo?

8

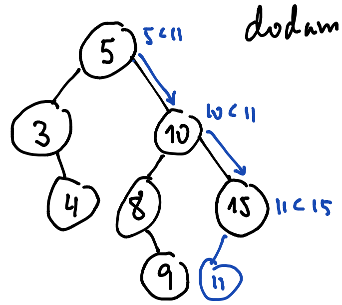


Iščemo x:

- → če je drevo prazno, x-a ni v drevesu
- → če je $x =$ korenu, smo našli
- → če je $x <$ koren, išči levo
- → če je koren $< x$, išči desno

Kako dodamo element?

dodamo 11

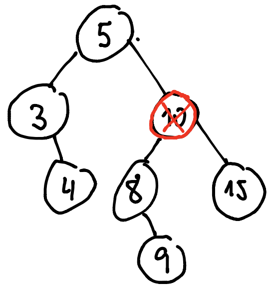


Dodamo tja, kjer bi moral biti, ko ga bomo iskali.

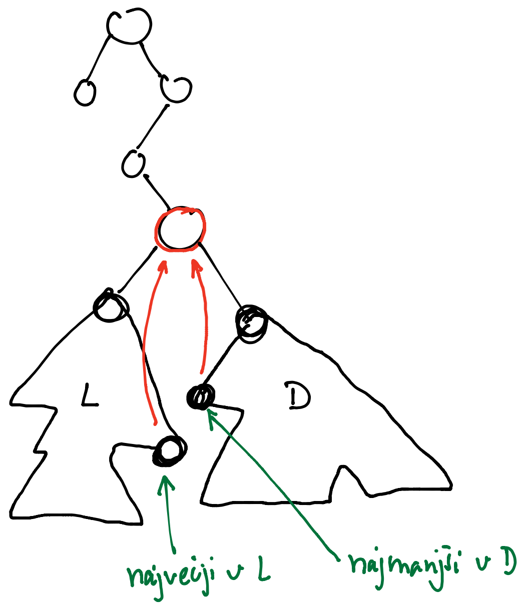
Dodaj x:

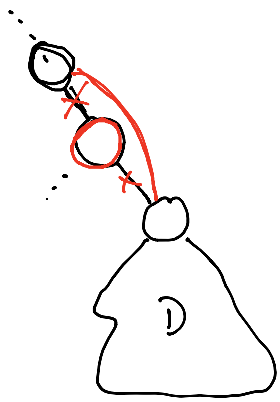
- → če drevo prazno, postavi x v koren
- → če $x =$ koren, je že v drevesu
- → če $x <$ koren, dodaj v levo poddrevo
- → če koren $< x$, dodaj v desno poddrevo

Kako odstranimo element?



odstranimo 10





Postopek:

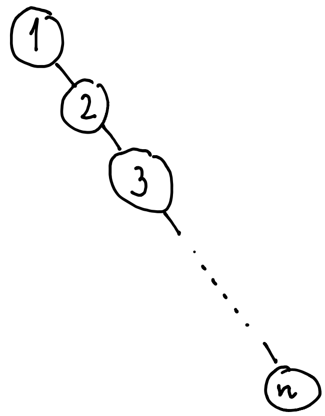
- poišči x (če ga nismo končali)
- nadomestimo x z najmanjšim v D ali z največjim v L

Časovna zahtevnost operacij

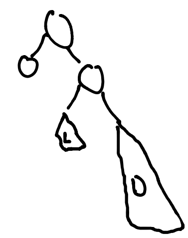
- iskanje : $O(\text{globina})$
- dodajanje : $O(\text{globina})$
- brisanje : $O(\text{globina})$

$\log_2 n \le \text{globina} \le n$

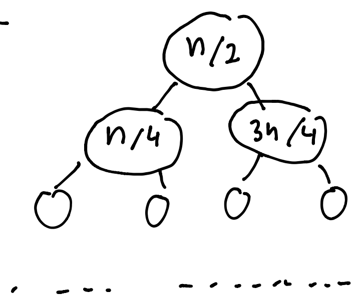
Ali je lahko iskalno drevo zelo globoko?

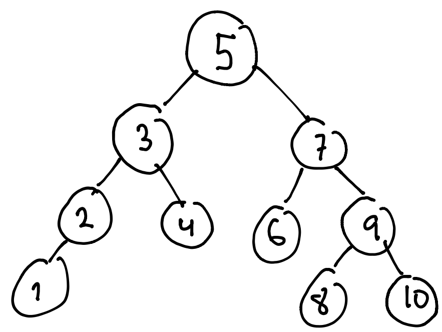


Tako drevo ni zaželeno.



<u>Plitko</u>:

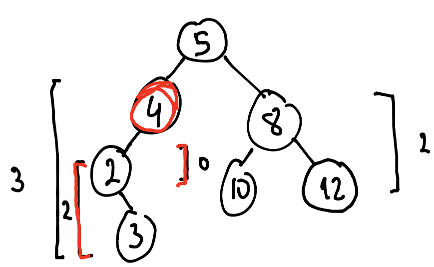




AVL drevesa

<u>Uravnoteženo drevo</u>: v <u>vsakem</u> vozlišču je razlika globin levega in desnega poddrevesa ≤ 1.

<u>Primer</u>:



Tu ni uravnoteženo

Izkaže se, da ima uravnoteženo drevo globino $O(\log_2 n)$

<u>AVL drevo</u> je uravnoteženo iskalno drevo.

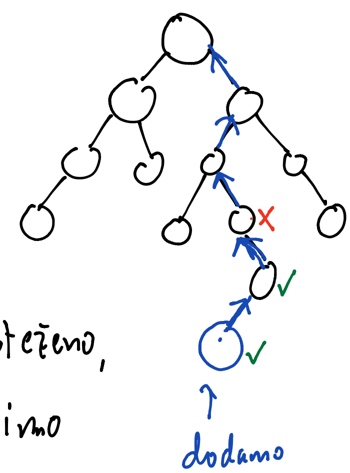
Poskrbimo, da dodajanje in brisanje elementov ohrani uravnoteženost.

<u>Dodajanje</u>:

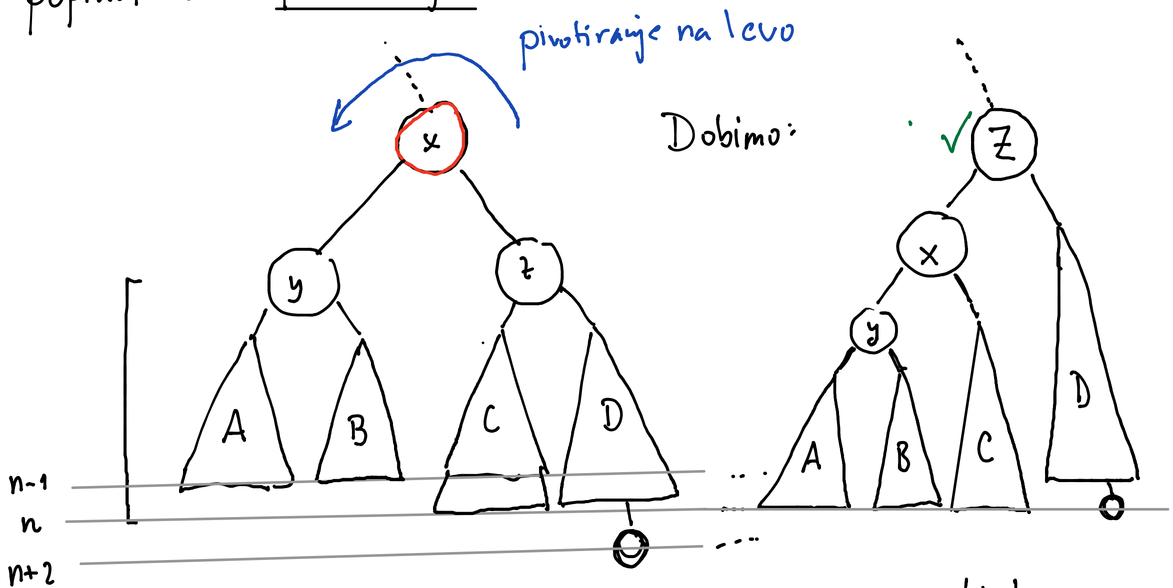
→ dodamo x, kjer bi ga iskali $O(\log n)$

→ popravimo uravnoteženost, če je to potrebno:

- od novega vozlišča potujemo navzgor proti korenu in preverjamo uravnoteženost
- ko najdemo vozlišče, ki ni uravnoteženo, ga popravimo (in s tem vzpostavimo uravnoteženost)



- Popravimo s <u>pivotiranjem</u>:



Druga varianta: ko postane C pregloboko

Globine dreves hranimo v vozliščih, da jih ni treba računati (ker to zahteva $O(n)$ korakov). Po potrebi popravimo globine, ko potujemo navzgor in ko pivotiramo.